Leader Election in Anonymous Rings: Franklin Goes Probabilistic

Rena Bakhshi[1], Wan Fokkink[1,2], Jun Pang[3], and Jaco van de Pol[4]

[1] Vrije Universiteit Amsterdam, Department of Computer Science
rbakhshi@few.vu.nl,wanf@cs.vu.nl
[2] CWI, Amsterdam
[3] Université du Luxembourg, Faculté des Sciences, de la Technologie et de la Communication
jun.pang@uni.lu
[4] University of Twente, Department of Computer Science
j.c.vandepol@ewi.utwente.nl

Abstract. We present a probabilistic leader election algorithm for anonymous, bidirectional, asynchronous rings. It is based on an algorithm from Franklin [22], augmented with random identity selection, hop counters to detect identity clashes, and round numbers modulo 2. As a result, the algorithm is finite-state, so that various model checking techniques can be employed to verify its correctness, that is, eventually a unique leader is elected with probability one. We also sketch a formal correctness proof of the algorithm for rings with arbitrary size.

1 Introduction

Leader election is the problem of electing a unique leader in a distributed network. It is required that all processes execute the same local algorithm.[1] Leader election is a fundamental problem in distributed computing and has numerous applications. For example, it is an important tool for breaking symmetry in a distributed system. Moreover, by choosing a process as the leader, it is possible to execute centralized algorithms in a decentralized environment. Leader election can also be used to recover from token loss for token-based algorithms, by making the leader responsible for generating a new token when the current one is lost.

There is a broad range of leader election algorithms. These algorithms vary in communication mechanism (asynchronous vs. synchronous), process names (unique identities vs. an anonymous network), network topology (e.g. ring, acyclic graph, complete graph). Here we focus on asynchronous communication with reliable channels but no message order preservation, and a bidirectional ring topology.

A classic leader election algorithm for unidirectional rings was given by Chang and Roberts [12]. It requires that each process has a unique identity,

[1] Else, the problem would be trivial: let one process perform the event "leader", while all other processes perform the event "not leader".

with a total ordering on identities; the process with the largest identity becomes the leader. The basic idea is that each process sends a message around the ring bearing its identity, where only the message with the largest identity completes the round trip. This algorithm requires $\mathcal{O}(n^2)$ messages in the worst case, but $\mathcal{O}(n \log n)$ on average. Franklin [22] developed a leader election algorithm for bidirectional rings with a worst-case message complexity of $\mathcal{O}(n \log n)$. The algorithm proceeds in election rounds, and each process is either active or passive. At the start of an election round, each active process sends its identity to its nearest active neighbours, and in return it receives such messages from these neighbours. An active process only progresses to the next election round if its own identity is larger than the two incoming identities. Peterson [41] and Dolev, Klawe and Rodeh [15] independently adapted Franklin's algorithm for unidirectional rings.

Sometimes the processes in a network cannot be distinguished by means of unique identities. Firstly, there is no concept of identity, e.g. Lego MindStorms robots. Secondly, as the number of processes in a network increases, it may become difficult to keep the identities of all processes distinct; or a network may accidentally assign the same identity to different processes. Thirdly, identities cannot always be sent around the network, for instance for reasons of efficiency; this is for instance the case in the leader election algorithm used within the IEEE 1394 (FireWire) standard, see [37]. In a so-called anonymous (or uniform) network, processes do not carry an identity. Angluin [1] showed that there does not exist a terminating deterministic algorithm for electing a leader in an anonymous, asynchronous network.

Itai and Rodeh [31, 32] studied how to break the symmetry in anonymous networks using probabilistic algorithms. They presented a probabilistic algorithm, based on the Chang-Roberts algorithm, to elect a leader in an anonymous unidirectional ring, under the assumption that all processes know the ring size.[2] At the start of an election round, active processes select a random identity from a finite domain, which they send around the ring. Active processes with the largest identity start a new election round if they detect a name clash, meaning that another process selected the same identity in the current round. Since the size of the ring is known, each process can recognise its own message by means of a hop counter, included in each message. The Itai-Rodeh algorithm terminates with probability one, and all its terminal states are correct, meaning that exactly one leader is elected. The average-case message complexity is $\mathcal{O}(n \log n)$.

In the Itai-Rodeh algorithm, an old message that has been overtaken by other messages in the ring, could in principle result in a situation where no leader is elected. To overcome this problem, successive election rounds are numbered, and each process and message is supplied with a round number. Thus an old message can be recognized and ignored. Fokkink and Pang [20, 21] showed that in case of FIFO channels, round numbers can be omitted from the Itai-Rodeh

[2] The latter assumption is essential; see e.g. [44, Sect. 9.4.1].

algorithm. They analysed the resulting algorithm using the probabilistic model checker PRISM [28].

In Sect. 2, we present a probabilistic leader election algorithm for anonymous bidirectional rings, based on Franklin's algorithm. As in the Itai-Rodeh algorithm, it is assumed that all processes know the ring size, and at the start of an election round, active processes select a random identity from a finite domain. We do not impose any assumption on the channel behaviour, i.e. the order of messages is not necessarily preserved between any pair of processes. Once again, each process can recognise its own message by means of a hop counter that is included in each message. However, instead of an infinite range of round numbers, we only need to keep track of round numbers modulo 2. This means that our probabilistic leader election algorithm is finite-state, and thus can be verified using explicit state space exploration (see Sect. 4). Furthermore, it implies that infinite executions, in which no leader is ever elected, violate "global fairness " (i.e., if in an infinite execution a transition from one global state of the system to another one $\gamma \rightarrow \gamma'$ can be taken infinitely often, then it is taken infinitely often); see Sect. 7.

We modelled our probabilistic version of Franklin's algorithm in the process algebraic language μCRL [8], and analysed for up to ring size six that a unique leader is elected. For ring size five, in case of a domain of three process identities, and for ring size six, in case of a domain of two process identities, we used the distributed version of the μCRL toolset [7] to store the generated state space over a cluster of computers. Moreover, we sketched a formal correctness proof for the algorithm in Sect. 4.

The model checker CADP [17] provided counter-examples to show that: (1) round numbers cannot be omitted from the probabilistic Franklin algorithm altogether (see Sect. 3), and (2) in case of a probabilistic version of the Dolev-Klawe-Rodeh algorithm, round numbers modulo 2 do not suffice (see Section 8). We used several optimizations, described in Sect. 5, to increase the efficiency of model checking of the probabilistic Franklin algorithm, notably confluence reduction. Moreover, using the probabilistic model checker PRISM, we made a performance comparison of two versions of the probabilistic Franklin algorithm: one in which fresh identities are chosen at the start of each election round, and one in which fresh identities are only chosen at the detection of an identity clash (see Sect. 6).

Related Work

Higham and Myers [27] present a leader election algorithm for anonymous, unidirectional rings of known size; their algorithm is similar to the algorithm of Itai and Rodeh, augmented with a time-out mechanism.

Fischer and Jiang [19] give a self-stabilizing leader election algorithm for anonymous, unidirectional rings, based on a *leader oracle* $\Omega?$, which for some point onwards is guaranteed to return the same leader to all processes (see also Sect. 7).

Several papers [11, 29, 33, 18] present leader election algorithms for anonymous rings of prime size, in the presence of a central demon, which acts as a scheduler.[3]

Leader election is related to token circulation for solving mutual exclusion problem, where having a token is interpreted as a permission to enter the critical section. A self-stabilizing token circulation algorithm guarantees eventual circulation of a unique token, even if the system is started from a global state where several tokens are present. Israeli and Jalfon [30] propose a self-stabilizing token circulation algorithm in an anonymous, bidirectional, asynchronous ring, in the presence of a centralized demon. In their algorithm, tokens move to the left or to the right with probability $\frac{1}{2}$, and merge when they meet, eventually reducing the number of tokens to one. However, without knowledge of ring size, the processes can never be sure whether a single token is left.

Mayer, Ofek, Ostrovsky and Yung [39] show that on an anonymous ring, leader election is equivalent to providing a self-stabilizing round-robin token management scheme. Angluin, Aspnes, Fischer and Jiang [2] give a self-stabilizing leader election algorithm for anonymous, unidirectional, asynchronous rings of unknown size. Beauquier, Gradinariu and Johnen [5] present a randomized self-stabilizing leader election algorithm under an arbitrary scheduler (no fairness assumption is required) on anonymous, unidirectional rings of known size, in the shared variables model. Both algorithms are based on token circulation.

Several other papers present self-stabilizing token circulation algorithms for anonymous, unidirectional rings: the algorithm of Herman [26] works on synchronous rings of odd size; Duchon, Hanusse and Tixeuil [16] present algorithms for synchronous rings of arbitrary size; Beauquier, Gradinariu and Johnen [4] and Datta, Gradinariu and Tixeuil [13] use several types of tokens and assume the synchronous communication model of shared variables, while the algorithm of Rosaz [42] uses the same idea in asynchronous message passing systems; the algorithms of Kakugawa and Yamashita [35] for asynchronous rings and Johnen [34] for shared memory settings run under unfair distributed schedulers. Of these papers, [4, 35, 34] require knowledge of ring size.

Mayer, Ostrovsky and Yung [40] give a randomized compiler for anonymous rings that transforms a self-stabilizing algorithm based on bidirectional communication to one that requires unidirectional, synchronous communication.

2 Franklin's Algorithm for Anonymous Rings

We consider a ring consisting of processes $p_0, \ldots, p_{n-1}$ for $n \geq 2$. Processes are anonymous, meaning that they do not carry a unique identity. Message-passing communication between processes is asynchronous, message order is

[3] Dijkstra [14] noted that such a leader election algorithm cannot exist if the ring size is a composite number.

not preserved between any pair of processes. Channels are bidirectional, so that a process p_i can send messages to its neighbours $p_{(i+1) \bmod n}$ and $p_{(i-1) \bmod n}$; a sent message is included in the message queue of its destination. It is assumed that receiving a message, processing it, and possibly sending a subsequent message take zero time. Channels are reliable, and the message queues are guided by a fair scheduler, meaning that every sent message will eventually be processed at its destination.

Each process is either active or passive. In our probabilistic version of Franklin's algorithm, an active process p_i maintains three parameters:

- $id_i \in \{1, \ldots, k\}$, for some $k \geq 2$, is its identity, not necessarily unique;
- $state_i$ ranges over $\{active, leader\}$;
- $bit_i \in \{T, F\}$ represents the number of the current election round modulo 2.

Passive processes simply pass on messages (increasing their hop counter by one).

All messages are of the form (*id*, *hop*, *bit*), travelling in both clockwise and counter-clockwise direction, where:

- *id* stores the identity of the process that originally sent the message;
- *bit* is a bit that represents the election round of this process modulo 2 (at the time that it sent the message);
- $hop \in \{1, \ldots, n\}$ is a counter, which initially has the value 1, and which is increased by one every time it is passed on by a process.

At the start of an election round, each active process p_i randomly selects an identity $id \in \{1, \ldots, k\}$, and sends a message with its identity to each of its two neighbours; initially, this message is of the form (id_i, 1, bit_i). Next, p_i receives such messages that originate from its two nearest active neighbours. Upon receipt of these messages, p_i determines whether it stays active for the next election round, by comparing three identities. If either of the messages it received has a larger identity than its own identity, then it becomes passive. Otherwise, it starts a new election round with a new identity. If a process gets a message with the hop counter equal to the network size n, the process becomes the leader ($state_i := leader$).

We now provide a more precise description of the algorithm. Initially, all processes p_i are active ($state_i = active$), and their bit bit_i is set to T.

– At the start of an election round with round number *bit*, an active process selects an identity $id \in \{1, \ldots, k\}$ and sends the message $(id, 1, bit)$ in both directions.
– Upon receipt of a message (id, hop, bit), a *passive* process passes on the message in the same direction, increasing the hop counter by one, i.e., $(id, hop + 1, bit)$.
– Upon receipt of a message (id, hop, bit) with $bit_i = bit$, an *active* process p_i executes the following steps:
  – if $hop = n$, then p_i becomes the leader ($state_i := leader$);
  – if $hop < n$, then p_i stores the message, and waits for a message with the bit bit_i from the opposite direction.

– An active process p_i stores messages that carry a bit $\neg bit_i$, to process them in the round with the appropriate bit.
– Upon receipt of messages with a bit bit_i from both directions, p_i checks whether either of these messages carries an identity larger than its own identity. If this is the case, then p_i becomes passive; otherwise, p_i starts a new election round, with an inverted bit as round number ($bit_i := \neg bit_i$) and a new identity.

3 Round Numbers Modulo 2 are Needed

Initially, we thought that our probabilistic version of the Franklin algorithm could maybe do without round numbers altogether. However, a model checking verification using the μCRL toolset [8] showed us that this is not true.[4] Fig. 3 shows a scenario where no leader is elected for a ring of size three and three identities. In this figure, black processes are active and white processes are passive.

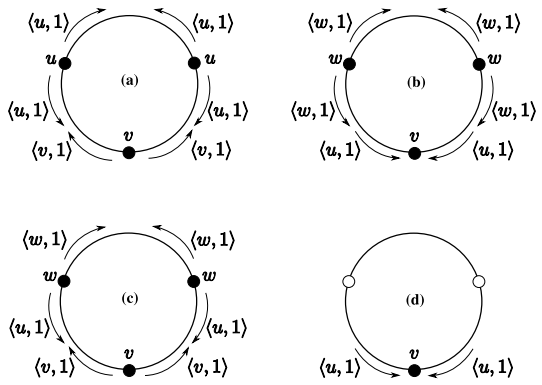

Fig. 1. Probabilistic Franklin algorithm without round numbers is flawed.

Initially, all processes are active; two processes select the same identity u, and one selects an identity $v < u$; all processes send a message with their identity in both directions (Fig. 3(a)). At the receipt of a message from both neighbours, the processes with identity u select a new identity $w < v$, and send messages carrying this new identity (Fig. 3(b)). The two messages $(u, 1)$ are overtaken by two messages with identity w. As $w < v$, process v proceeds to a next election round, in which it selects the identity v again, and sends messages $(v, 1)$ in both direction (Fig. 3(c)). Upon the receipt of messages $(v, 1)$ and $(w, 1)$, the processes with identity w become passive (Fig. 3(d)). Finally, the

[4] Lamport [36] actually advocates that all distributed algorithms should be model checked before publication.

outdated messages $(u, 1)$ make the process with identity v passive as well; all processes have become passive now.

4 Correctness Analysis

We say that an execution of the algorithm has *terminated* if each process is either passive or elected as the leader, and there are no remaining messages in the channels. We argue that the probabilistic Franklin algorithm for anonymous bidirectional rings terminates with probability one, and upon termination a unique leader has been elected.

For a start, we modelled the probabilistic Franklin algorithm with round numbers modulo 2 in the μCRL framework [8], with channels that have an unbounded capacity, each implemented as a buffer, and its correctness has been verified for rings with a size up to six processes. The input language for μCRL is based on process algebra and abstract data types. Our μCRL specification is available at [3]. Tables 1(a) and 1(b) provide state space generation results for domains of two and three identities, respectively. To carry out the verification for six processes (in case of two identities) and five processes (in case of three identities), a distributed version of the μCRL toolset was used. The resulting state space was reduced using branching bisimulation equivalence [46, 25], which eliminates internal and communication transitions (i.e., only "leader" transitions are not abstracted away), while maintaining the branching structure of the state space. In case of the distributed version of the μCRL toolset, we applied a distributed reduction algorithm from [9].

Table 1. State space generation statistics

(a) State space for two identities

# Procs	States	Transitions
2	657	1,368
3	15,445	43,968
4	380,609	1,396,512
5	9,819,065	44,242,920
6	260,753,105	1,393,967,976

(b) State space for three identities

# Procs	States	Transitions
2	1,525	3,564
3	55,009	168,102
4	2,095,777	8,182,092
5	84,381,157	401,681,445

The μCRL specification language does not allow to express probabilities. Still we could verify that although there are infinite executions, with probability one, eventually always a leader is elected. This is because branching bisimulation equivalence abstracts away from infinite executions that violate global fairness. That is, after minimization modulo this equivalence, such executions have been eliminated. The minimized state space of our algorithm consisted of only two states s_1 and s_2, where the initial state s_1 can perform a leader action to s_2, which is a terminated state.

We now sketch a formal correctness proof of the probabilistic Franklin algorithm.

Proposition 1. *If channels are FIFO, the probabilistic Franklin algorithm behaves correctly, even if processes and messages do not keep track of round numbers at all. That is, upon termination, exactly one leader has been elected.*

Proof. In case of FIFO channels, it is guaranteed that in each election round, an active process always receives messages from the left and the right that were created in this election round (cf. [20, 21]). Therefore round numbers are redundant.

In each election round, active processes with the largest identity in that round do not become passive. And an active process can only become the leader if all other processes have become passive. From this it follows that upon termination there is a unique leader. □

We now focus on showing that in the probabilistic Franklin algorithm, round numbers modulo 2 suffice to enforce FIFO behaviour of these channels.

Lemma 1. *After initialization, and before a leader is elected, the following invariant holds for the algorithm. Between each pair of active processes p, p' there are exactly two messages m, m'.*

- *If m, m' travel in opposite directions, p, p', m, m' all carry the same bit as round number.*
- *If m, m' travel in the same direction, p, p' have opposite bits, and m, m' have opposite bits.*

Proof. Fig. 2 depicts three cases (a symmetric variant of Fig. 2(a) is omitted) consisting of a triple of adjacent active processes, wherein the middle process with the bit $b \in \{T, F\}$ receives two incoming messages from its neighbours (Figs. 2(a), 2(d), 2(g)). If neither of the messages it received has a larger identity than its own identity, then it starts a new election round with the bit $\neg b$ (Figs. 2(b), 2(e), 2(h)). Otherwise, it becomes passive (Figs. 2(c), 2(f), 2(i)). In all six cases, the invariant holds. □

Theorem 1. *In the probabilistic Franklin algorithm (with reliable, but not necessarily FIFO channels), upon termination, exactly one leader has been elected.*

Proof. From the invariant in Lemma 1 it follows that in the probabilistic Franklin algorithm with round numbers modulo 2, channels behave as FIFO queues. Namely, if there are two messages travelling to an active process in the same direction, they have opposite bits. So the active process can recognize which of these two messages was created in its current election round. Hence, the theorem follows from Prop. 1. □

Theorem 2. *The probabilistic Franklin algorithm terminates with probability one.*

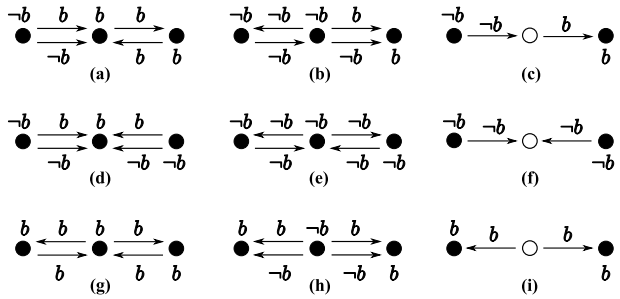

Fig. 2. Illustration of the invariant

Proof. When there are $\ell \geq 2$ active processes in the ring, these processes all remain active only if they all the time choose the same identity. Otherwise, at least one active process will become passive. The probability that all active processes select the same identity in one election round is $(\frac{1}{k})^{\ell-1}$, where k is the number of possible identities. Thus, the probability for all ℓ active processes to choose the same identity m times in a row is $(\frac{1}{k})^{m(\ell-1)}$. As $k \geq 2$, the probability that the number of active processes eventually decreases is one.

On average, the probabilistic Franklin algorithm takes $\mathcal{O}(n \log n)$ messages to terminate. (On average, in each election round about $\frac{3}{8}$ of the active processes become passive, so there are in the order of $\log n$ rounds; and each election round takes $2n - 2$ messages.)

5 Optimisation Techniques for Generating the State Space

To obtain a smaller state space, we simplified the algorithm described in Sect. 2: every round a node could decide to read always first a message from the left neighbour, and then from the right neighbour. This does not really influence the behaviour of the algorithm, because a node only take visible actions after receiving a message from both sides. We modelled this version of the algorithm in the μCRL toolset, and verified it up to six processes[5] in the following manner.

First the parallel operators are eliminated by the linearisation algorithm from [24]. Next it is symbolically reduced by static analysis: constant propagation (replace provably constant parameters by their initial value) [23], and dead variable analysis (reset variables that are not used anymore to a default value). The number of states and transitions of the state space that have been generated in this way are presented as the "normal" strategy in Tables 2 and 3.

[5] A distributed version of the μCRL toolset [7] was used for six processes (in case of two identities) and five processes (in case of three identities) with the "normal" state space generation strategy.

For more efficient state space generation, we applied symbolic confluence reduction [10]. To this end, a theorem prover can be used to automatically detect and mark confluent τ's, i.e. internal transitions and hidden communications that are not causally related, for instance, because they occur at different parallel components. Confluence can then be exploited on a symbolic level by giving priority to confluent τ's, marked by the theorem prover. This reduction keeps only the confluent τ's going out of a state, and all the other transitions going out of the state are removed. This symbolic prioritization is implemented in the Confelm tool [6] from μCRL. We used it to remove confluent τ-summands, marked by the theorem prover Confcheck [45] from μCRL.

We also experimented with an on-the-fly τ-reduction [10, 38]. It is based on Tarjan's algorithm for decomposition of a graph into its strongly connected components [43]. In this reduction, for each state a representative state is computed, which it can reach by means of confluent τ-transitions. To compute the representative of a state, a depth-first search traversal via the confluent τ-transitions is made, until a state with a known representative is encountered, or a 'terminal' strongly connected component of confluent τ-transitions is found. (Terminal means that there are no outgoing confluent τ-transitions.) In the former case the known representative is returned, and in the latter case the state where the terminal strongly connected component was entered is returned. In the state space generation algorithm from [6], only representatives of states are generated.

After generation of the (partially reduced) state space, we performed a full reduction modulo branching bisimulation. The resulting state space consisted of only two states s_1 and s_2, where the initial state s_1 can perform a leader action to s_2, which is a terminated state.

Strategy	# Proc.	2	3	4	5	6
normal	s.	385	7,613	152,065	3,162,337	67,758,817
	t.	664	17,880	459,488	11,736,100	298,484,184
confelm	s.	205	2,875	40,881	606,783	9,280,633
	t.	340	6,342	114,384	2,069,040	37,381,488
confelm	ext. s.	165	1,819	21,409	263,963	3,348,345
+	int. s.	181	2,343	30,039	395,723	5,350,021
on-the-fly	t.	276	4,086	60,576	902,820	13,449,324

Table 2. State space for the probabilistic Franklin algorithm with 2 identities.

Tables 2 and 3 show state space generation results of the simplified algorithm (states and transitions) for domains of two and three identities, respectively, with the different reduction strategies. For the on-the-fly τ-reduction, the number of states generated in the end (external states) differs from the number of states that are internally computed (internal states).

Strategy	# Proc.	2	3	4	5
normal	s.	877	26,299	802,489	25,919,965
	t.	1,680	65,853	2,560,848	100,868,445
confelm	s.	469	9,874	214,957	4,952,449
	t.	876	23,310	637,884	17,778,660
confelm	ext. s.	385	6,400	116,785	2,242,609
+	int. s.	433	8,518	170,131	3,524,305
on-the-fly	t.	732	15,570	353,508	8,137,080

Table 3. State space for the probabilistic Franklin algorithm with 3 identities

6 Performance Comparison with PRISM

In Sect. 2, we presented the probabilistic Franklin algorithm in which an active process chooses a fresh identity at the start of each election round (Algorithm A). There is one variant of this algorithm (Algorithm B) in which an active process only chooses a fresh identity at the start of a new election round if either of the two messages it received in the previous election round carried an identity equal to its own identity.

The probabilistic model checker PRISM [28] has the ability to automatically compute precise quantitative results based on exhaustive analysis of a formal model. For both versions, we used PRISM version 3.1.1 to calculate the probabilities of electing a unique leader within t "discrete time steps" (up to 150), where each such step corresponds to one transition in the algorithm. The experimental results presented in Fig. 3 indicate that Algorithm A has a much better performance than Algorithm B. Note that when t moves to infinity, both algorithms elect a unique leader with probability one.

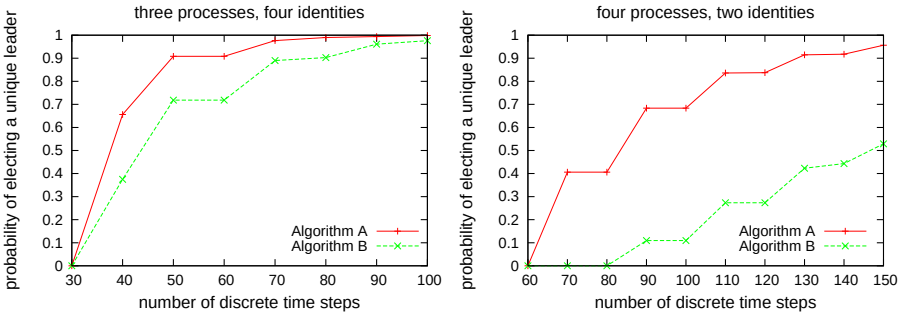

Fig. 3. The probability of electing a unique leader with deadlines.

7 Global Fairness

Fischer and Jiang [19] give a leader election algorithm for anonymous, unidirectional rings, without requiring global knowledge of ring size. Instead, they require a *leader oracle* $\Omega?$, which can by each process be asked who is the leader,

and which for some point onwards is guaranteed to return the same answer to all processes. Under the assumption of what they call global fairness (i.e., if in an infinite execution a transition from one global state of the system to another one $\gamma \rightarrow \gamma'$ can be taken infinitely often, then it is taken infinitely often), they prove that their algorithm always terminates successfully.

Fischer and Jiang [19, p403] write: "We leave open the question of whether such an algorithm exists without the help of Ω?." Actually, in the absence of global knowledge of ring size, it is straightforward to provide a negative answer to this question. Namely, if such a leader election algorithm existed, then upon successful termination, the leader could start a traversal to determine the correct ring size. However, for anonymous rings, each ring size computation algorithm has a positive probability of computing the wrong ring size (see [44, Sect. 9.4.1]).

On the other hand, under the assumption of global knowledge of ring size, our probabilistic version of Franklin's algorithm provides a positive answer to the question of Fischer and Jiang, in the case of bidirectional rings. Namely, owing to the fact that our algorithm is finite-state, each globally fair infinite execution should at some point reach a configuration in which one active process selects a larger identity than all other active processes, meaning that the execution will terminate with this process as leader. But this contradicts with the fact that the execution is infinite. In other words, in our algorithm each infinite execution is not globally fair.

We note that this argumentation does not apply to the Itai-Rodeh algorithm, due to the presence of an infinite range of round numbers. As a consequence, in that algorithm no infinite execution visits a configuration infinitely often.

8 Probabilistic Dolev-Klawe-Rodeh Algorithm

The Dolev-Klawe-Rodeh algorithm is an adaptation of Franklin's algorithm to unidirectional rings. In an election round, an active process p compares its identity with the identities of the two closest active processes on its left. The process p proceeds to the next election round only if the identity of the closest active process on its left is the largest of these three identities. A natural question is whether the idea of round numbers modulo 2 would also apply to that algorithm. We therefore modelled a probabilistic version of the Dolev-Klawe-Rodeh algorithm with round numbers modulo 2 in μCRL. We detected by a model checking analysis using μCRL and CADP toolsets that this algorithm is flawed, in the sense that no leader may be elected. This is due to the fact that in the probabilistic Dolev-Klawe-Rodeh algorithm, round numbers modulo 2 do not enforce FIFO behaviour of channels. This is depicted in Fig. 4.

In Fig. 4, a scenario is depicted in which a message $\langle 2, two, 2, T\rangle$ is overtaken by a newer message $\langle 0, two, 2, T\rangle$. In this picture, processes carry a round number modulo 2 (T or F). Moreover, messages carry a value (the first parameter), a hop counter (the third parameter), and a round number modulo 2 (the fourth parameter). The second parameter in a message, *one* or *two*, keeps track

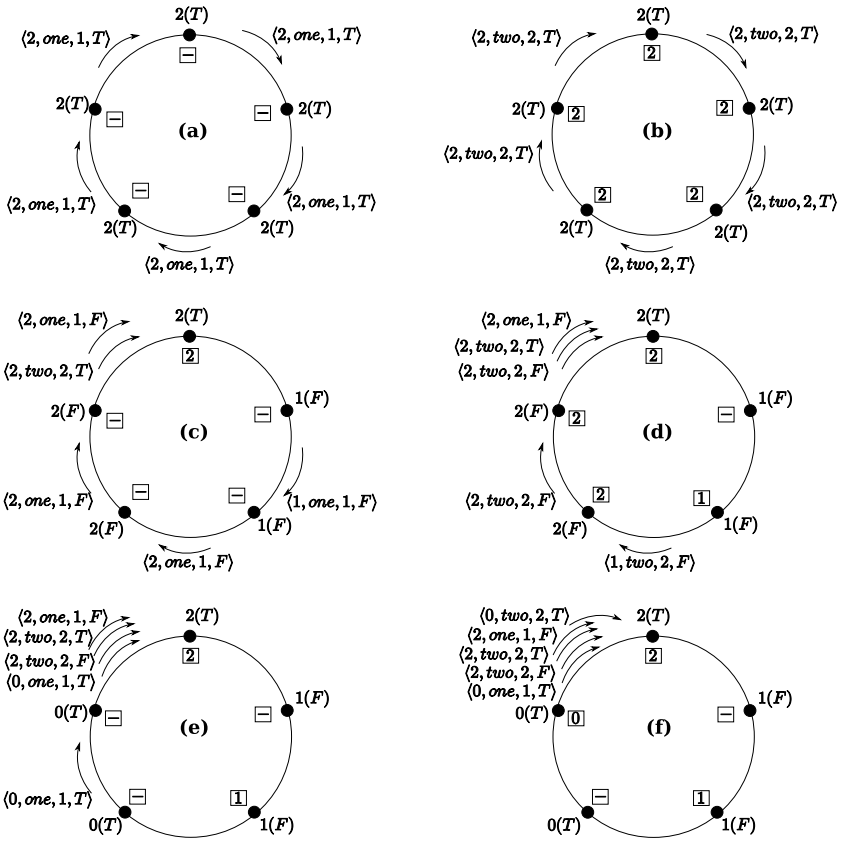

Fig. 4. Probabilistic Dolev-Klawe-Rodeh with round numbers modulo 2 is flawed.

whether a message is travelling from its originator to the next active process, or has been forwarded by an active process, respectively.

Acknowledgement

We thank Bert Lisser for performing the experiments on the distributed version of the μCRL toolset.

References

1. D. Angluin. Local and global properties in networks of processors. In *Proc. 12th ACM Symp. on Theory of Computing*, pages 82–93. ACM, 1980.
2. D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang. Self-stabilizing population protocols. In *Proc. 9th Conf. on Principles of Distributed Systems*, volume 3974 of *LNCS*, pages 103–117. Springer, 2005.

3. R. Bakhshi, W. Fokkink, J. Pang, and J. van de Pol. μCRL specification of probabilistic Franklin leader election algorithm. `http://www.few.vu.nl/~rbakhshi/alg/franklin.mcrl`.
4. J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *Proc. 18th Symp. on Principles of Distributed Computing*, pages 199–207. ACM, 1999.
5. J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. *Distributed Computing*, 20(1):75–93, 2007.
6. S. Blom. Partial τ-confluence for efficient state space generation. Technical Report SEN-R0123, CWI, Amsterdam, The Netherlands, 2001.
7. S. Blom, J. Calamé, B. Lisser, S. Orzan, J. Pang, J. van de Pol, M. Torabi Dashti, and A. Wijs. Distributed analysis with μCRL: A compendium of case studies. In *Proc. 13th Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 683–689. Springer, 2007.
8. S. Blom, W. Fokkink, J.-F. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *Proc. 13th Conf. on Computer Aided Verification*, volume 2102 of *LNCS*, pages 250–254. Springer, 2001.
9. S. Blom and S. Orzan. Distributed branching bisimulation reduction of state spaces. In *Proc. 2nd Workshop on Parallel and Distributed Model Checking*, volume 89(1) of *ENTCS*. Elsevier, 2003.
10. S. Blom and J. van de Pol. State space reduction by proving confluence. In *Proc. 14th Conf. on Computer Aided Verification*, volume 2404 of *LNCS*, pages 596–609. Springer, 2002.
11. J. Burns and J. Pachl. Uniform self-stabilizing rings. *ACM Trans. Program. Lang. Systems*, 11(2):330–344, 1989.
12. E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283, 1979.
13. A. Kumar Datta, M. Gradinariu, and S. Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *Proc. 14th Int. Parallel & Distributed Processing Symp.*, pages 465–470. IEEE Computer Society, 2000.
14. E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
15. D. Dolev, M. Klawe, and M. Rodeh. An $O(n \log n)$ unidirectional algorithm for extrema finding in a circle. *J. of Algorithms*, 3(3):245–260, 1982.
16. P. Duchon, N. Hanusse, and S. Tixeuil. Optimal randomized self-stabilizing mutual exclusion in synchronous rings. In *Proc. 18th Symp. on Distributed Computing*, volume 3274 of *LNCS*, pages 216–229. Springer Verlag, 2004.
17. J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP - a protocol validation and verification toolbox. In *Proc. 8th Conf. on Computer Aided Verification*, volume 1102 of *LNCS*, pages 437–440. Springer, 1996.
18. F. Fich and C. Johnen. A space optimal, deterministic, self-stabilizing, leader election algorithm for unidirectional rings. In *Proc. 15th Conf. on Distributed Computing*, volume 2180 of *LNCS*, pages 224–239. Springer, 2001.
19. M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Proc. 10th Conf. on Principles of Distributed Systems*, volume 4305 of *LNCS*, pages 395–409. Springer, 2006.

20. W. Fokkink and J. Pang. Simplifying Itai-Rodeh leader election for anonymous rings. In *Proc. 4th Workshop on Automated Verification of Critical Systems*, volume 128(6) of *ENTCS*, pages 53–68. Elsevier, 2005.
21. W. Fokkink and J. Pang. Variations on Itai-Rodeh leader election for anonymous rings and their analysis in PRISM. *J. of Universal Computer Science*, 12(8):981–1006, 2006.
22. R. Franklin. On an improved algorithm for decentralized extrema finding in circular configurations of processors. *Commun. ACM*, 25(5):336–337, 1982.
23. J. F. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. *SIGPLAN Notices*, 37(12):98–107, 2002.
24. J.-F. Groote, A. Ponse, and Y. Usenko. Linearization in parallel pcrl. *J. Log. Algebr. Program.*, 48(1-2):39–70, 2001.
25. J.-F. Groote and F. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proc. 17th Colloq. on Automata, Languages and Programming*, volume 443 of *LNCS*, pages 626–638. Springer, 1990.
26. T. Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.
27. L. Higham and S. Myers. Self-stabilizing token circulation on anonymous message passing. In *Proc. 2nd Conf. on Principles of Distributed Systems*, pages 115–128. Hermes, 1998.
28. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
29. S.-T. Huang. Leader election in uniform rings. *ACM Trans. Program. Lang. Systems*, 15(3):563–573, 1993.
30. A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 119–131. ACM, 1990.
31. A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In *Proc. 22nd Symp. on Foundations of Computer Science*, pages 150–158. IEEE, 1981.
32. A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, 1990.
33. G. Itkis, C. Lin, and J. Simon. Deterministic, constant space, self-stabilizing leader election on uniform rings. In *Proc. 9th Workshop on Distributed Algorithms*, volume 972 of *LNCS*, pages 288–302. Springer, 1995.
34. C. Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings. In *Proc. 21st Symp. on Reliable Distributed Systems*, pages 80–89. IEEE Computer Society, 2002.
35. H. Kakugawa and M. Yamashita. Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. *J. Parallel Distrib. Comput.*, 62(5):885–898, 2002.
36. L. Lamport. Checking a multithreaded algorithm with $^{+}$CAL. In *Proc. 20th Symp. on Distributed Computing*, volume 4167 of *LNCS*, pages 151–163. Springer, 2006.
37. S. Maharaj and C. Shankland. A survey of formal methods applied to leader election in IEEE 1394. *J. of Universal Computer Science*, 6(11):1145–1163, 2000.
38. R. Mateescu. On-the-fly state space reductions for weak equivalences. In *Proc. 10th Workshop on Formal Methods for Industrial Critical Systems*, pages 80–89. ACM, 2005.

39. A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung. Self-stabilizing symmetry breaking in constant-space. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 667–678. ACM, 1992.
40. A. Mayer, R. Ostrovsky, and M. Yung. Self-stabilizing algorithms for synchronous unidirectional rings. In *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms*, pages 564–573. Society for Industrial and Applied Mathematics, 1996.
41. G. Peterson. An $O(n \log n)$ unidirectional algorithm for the circular extrema problem. *ACM Trans. Program. Lang. Systems*, 4(4):758–762, 1982.
42. L. Rosaz. Self-stabilizing token circulation on asynchronous uniform unidirectional rings. In *Proc. 19th ACM Symp. on Principles of Distributed Computing*, pages 249–258. ACM, 2000.
43. R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. on Computing*, 1(2):146–160, 1972.
44. G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000. 2nd edition.
45. J. van de Pol. A prover for the μCRL toolset with applications, version 0.1. Technical Report SEN-R0106, CWI, Amsterdam, The Netherlands, 2001.
46. R. van Glabbeek and P. Weijland. Branching time and abstraction in bisimulation semantics. *J. of the ACM*, 43(3):555–600, 1996.